Tampere University

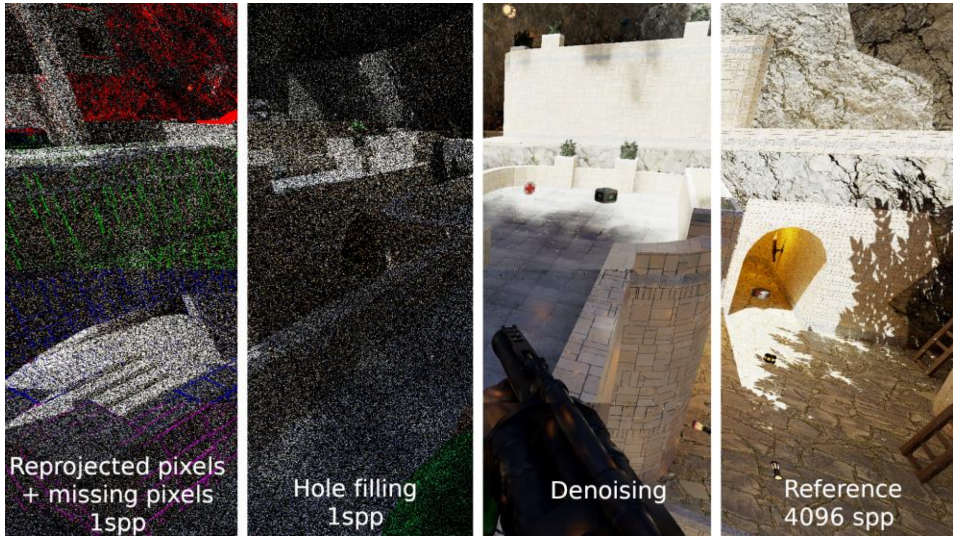
(Read the full paper)

Real-time Stereoscopic Image-parallel Path tracing

Erwan Leria, Markku Mäkitalo, Pekka Jääskeläinen

Tampere University, Finland

(erwan.leria@tuni.fi)



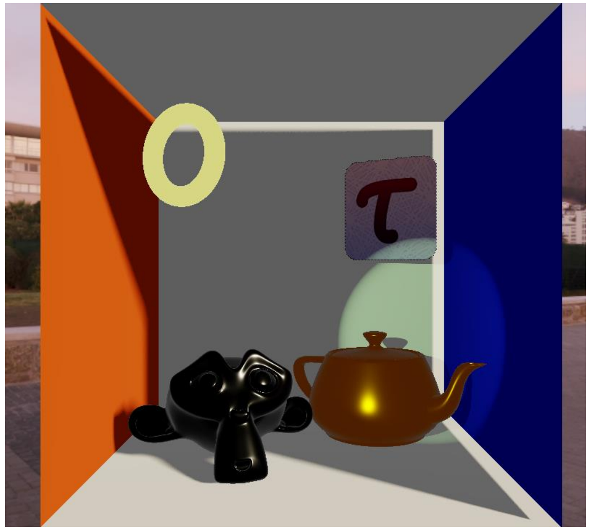
imaging.org
electronic IMAGING 2024 JANUARY 21-25

Motivation

- **Global illumination effects** provide better inference about the surrounding objects (Murray, 2019)
- Rasterization only renders a subset of those global illumination effects and mimicking them requires manual hacks
- Ray-tracing techniques such as path tracing provides all those global illumination effects

Path tracing

Rasterization



Motivation

- Ray tracing techniques are computationally expensive for a single GPU in real-time
- Extend stereoscopic ray tracing techniques to a single multi-GPU node
- Reach real-time photorealistic rendering with virtual reality

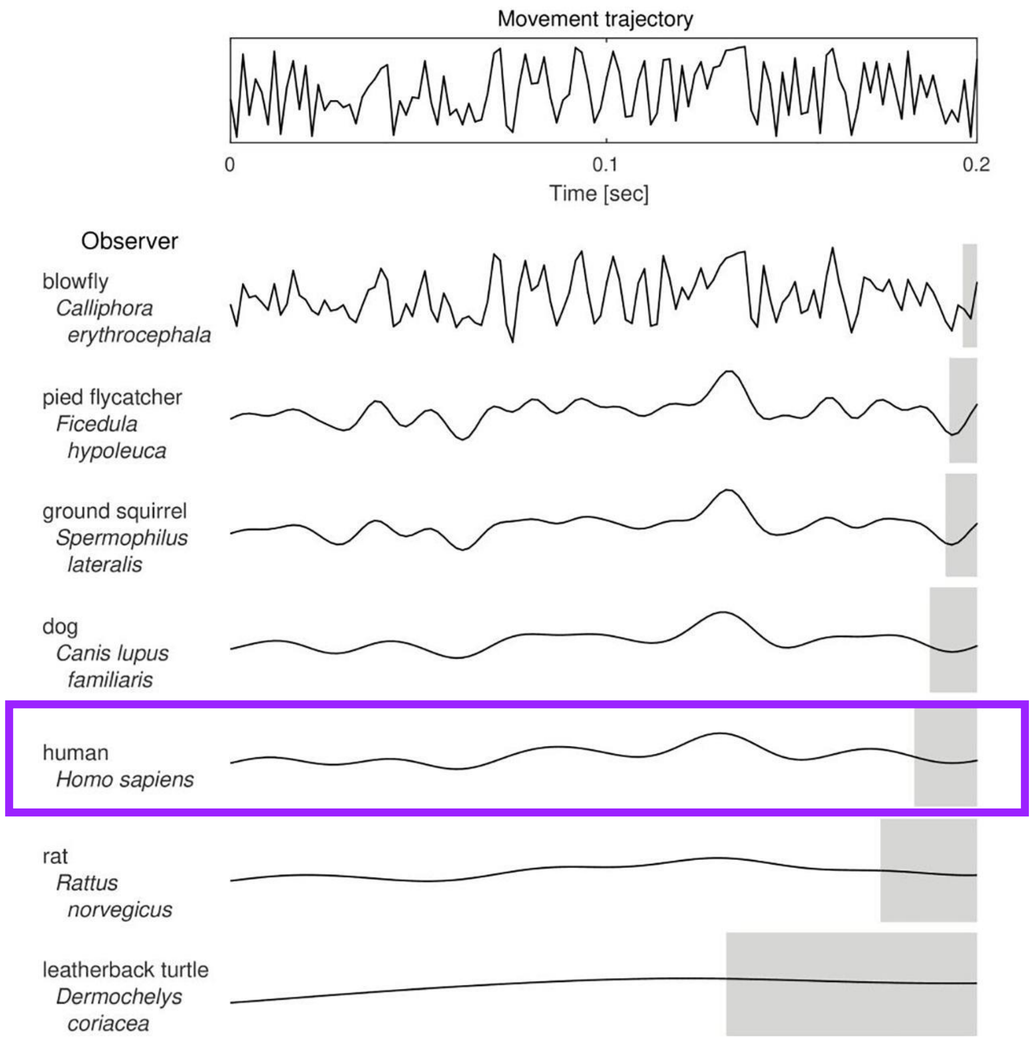
(Ouyang, 2021)

Real-time constraint

- **Avoid cybersickness** (nausea, dizziness):
  - User must not notice any lag between consecutive images

- We set the **motion to photon (end to end) latency** to be between **11-20 ms**
  - Based on the human **Critical Flicker Frequency** (CFF) range: **50-90 Hz** (Mankowska, 2021)

- We do not consider eccentricity or contrast sensitivity for the CFF



Example: Movement detection of a blowfly
(Jura, 2019)

Background

- Parallel rendering pipelines categories (Molnar, 1994)
  - The scene or the image is divided into multiple parts that are assigned to GPUs

Data-parallel
(Scene partitioning)

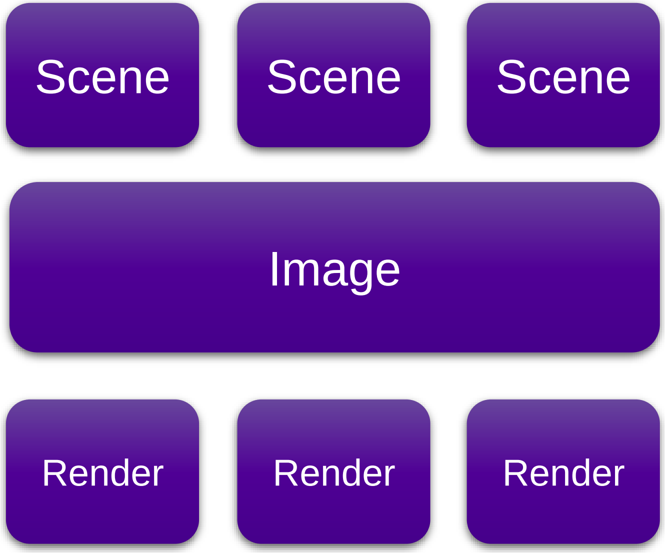
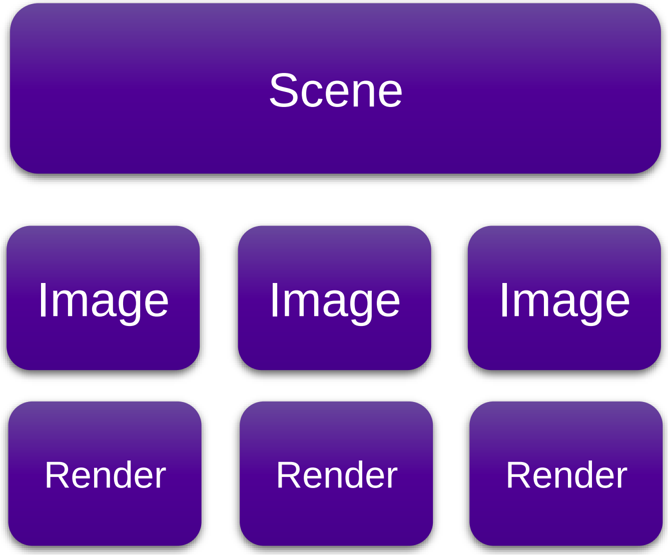
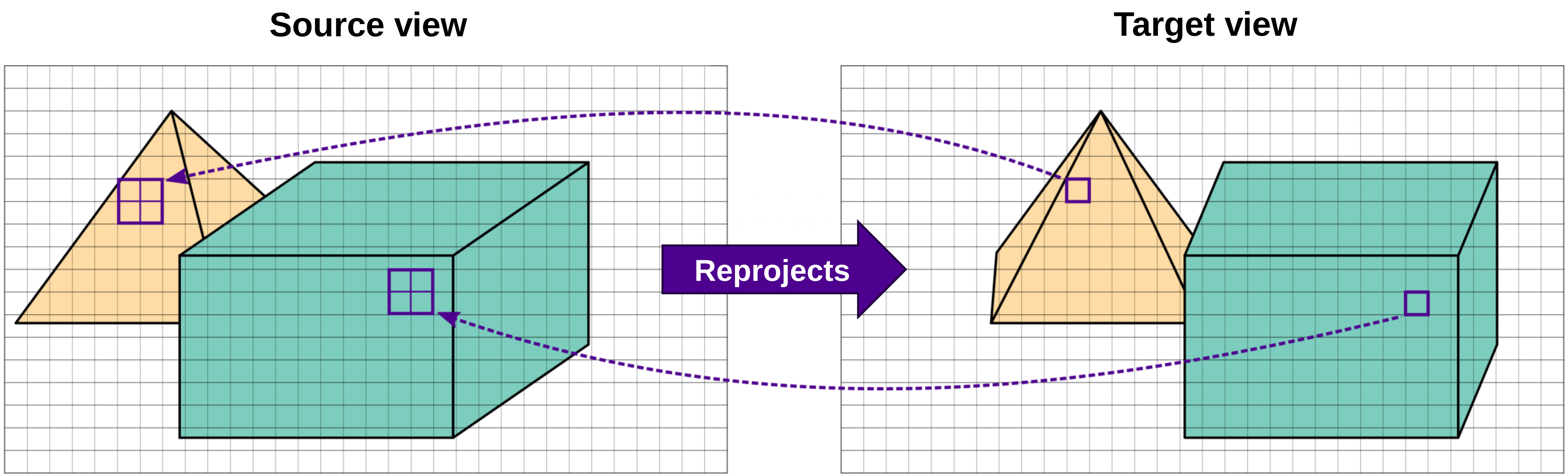


Image-parallel
(Image-space subdivision)



Background

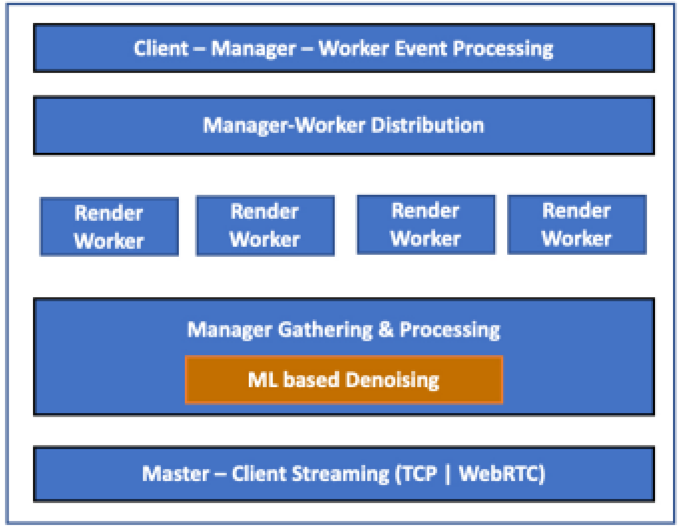
- **Spatial reprojection** (Adelson, 1993)
  - Reprojects pixels from a source view to a target view
  - **Reduce computational cost** of rendering both views
  - Reprojectable target pixels are rendered with bilinear filtering from source pixels



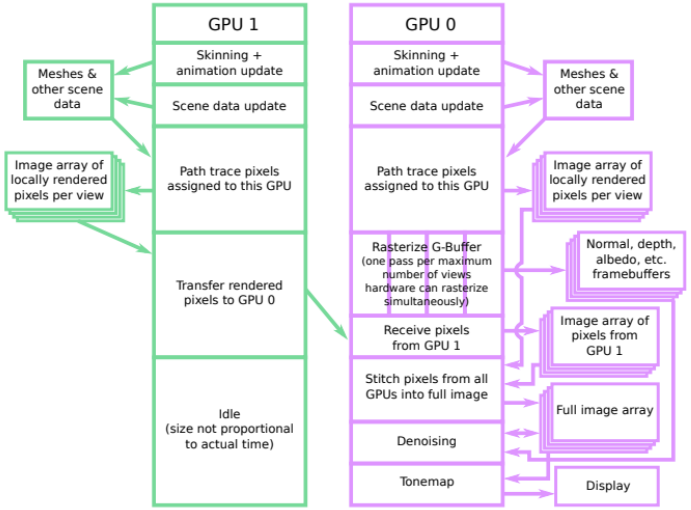
Related works

1. Facebook/Meta Reality Lab Cycles (F. Xie, 2021)
2. Tauray (Ikkala, 2022) - **baseline that we are using**
3. OO-VR (C. Xie, 2019)

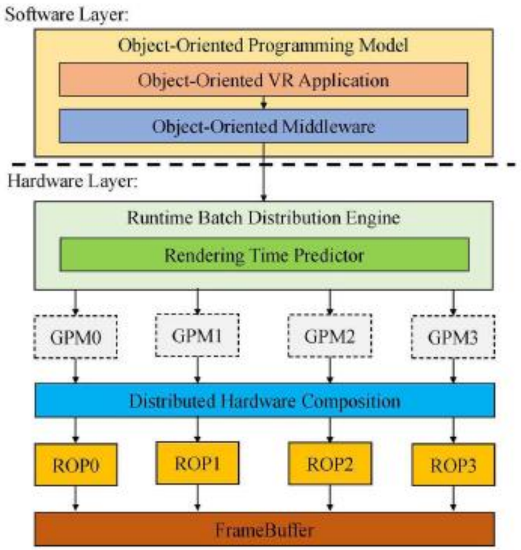
1



2



3



Technical challenges

- Related work pitfalls:
  - GPUs: only used for rendering (rasterization/path tracing)
  - Main GPU: assumes post-processing tasks alone

- **Goal**:
  - Extend spatial reprojection and post-processing algorithms to multiple GPUs
    - Adds dependencies in the pipeline between stages and between views
  - Handle quality--performance awareness for scene variability

Proposed rendering pipeline

Overview of the proposed pipeline

Parallelized to multiple GPUs

- 0. Skinning
- 1. Scene update
- 2. Path tracing
- 3. G-buffer rasterization
- 4. Spatial reprojection
- 5. Hole-filling
- 6. Denoising
- 7. Record prev. buffers
- 8. Tonemap.
- 9. Transfers
- 10. Stitching
- 11. Blit

Image-space subdivision (stage #2)

- GPUs path trace their image region in the **left** view
- Large **rectangular** regions: **maximize data locality** along the **horizontal axis**

Left view (source viewport)

GPU 0

GPU 1

GPU 2

GPU 3

Subdivision algorithm: *subdivision of the image into N rectangular regions, each assigned to a GPU*

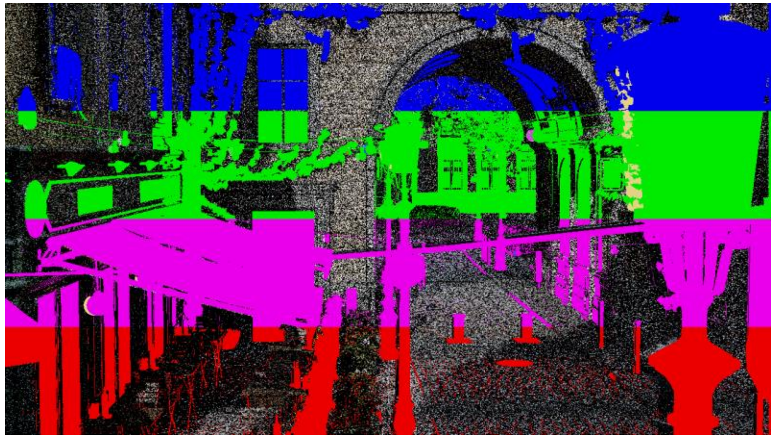
Reprojected right view (stage #4)

- Spatial reprojection: **read** pixels in the **source** and **reproject** them in the **target**
- **Discarded pixels = non-reprojectable pixels**

Left view (source viewport)

Reproject pixels

Right view (target viewport)



GPU
GPU 0
GPU 1
GPU 2
GPU 3

Discarded pixels
Discarded pixels in GPU 0
Discarded pixels in GPU 1
Discarded pixels in GPU 2
Discarded pixels in GPU 3

Hole-filling (stage #5)

- Additional pass of path tracing only for the **right** view in the holes

Left view (source viewport)	Right view (target viewport)
	
GPU 0	GPU 0
GPU 1	GPU 1
GPU 2	GPU 2
GPU 3	GPU 3

Tampere University

Denoising (stage #6)

- Applying SVGF denoiser on **both** views

Left view (source viewport)

GPU 0
GPU 1
GPU 2
GPU 3

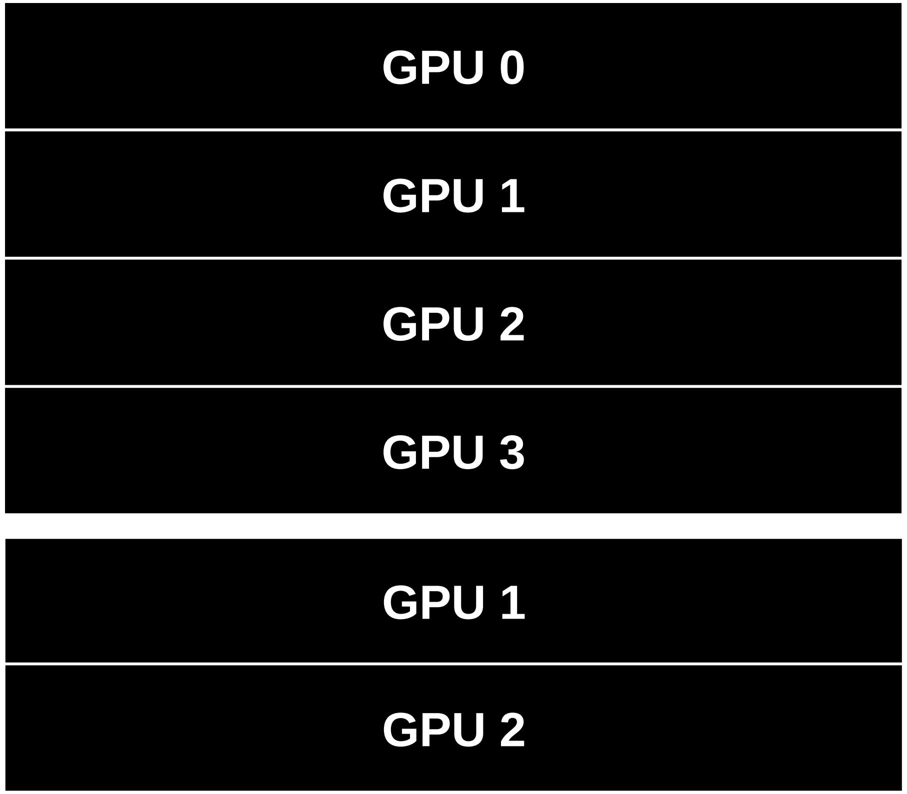
Right view (target viewport)

GPU 0
GPU 1
GPU 2
GPU 3

Denoising (stage #6)

- **Seam artifacts** (**difference of contrast**) due to denoising with **spatial loss of information** at the border **between two GPUs** image regions

Seam artifacts



Task scheduling framework

- Manage dependencies between the stages

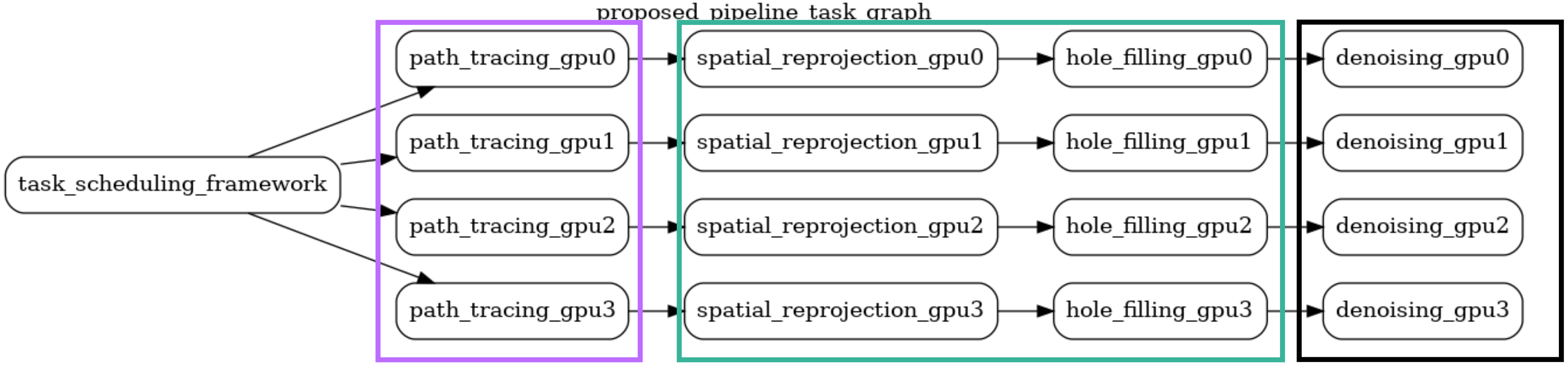
(Main stages)

2. Path tracing

4. Spatial reprojection

5. Hole-filling

6. Denoising



Left view

Right view

Both views

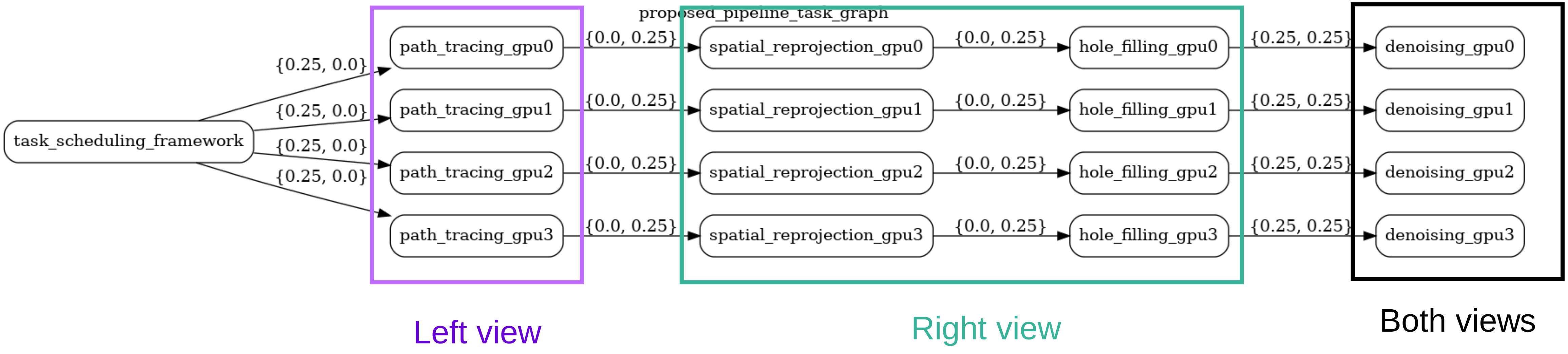
Task scheduling framework

- Not all the stages operate on all the views
- We need to define the number of pixels processed per GPU per view per stage
- Workload ratio: ratio of pixels processed at a given stage

- Given a GPU i, we define $w_i = \{w_l, w_r\}$**, for the left and right view**

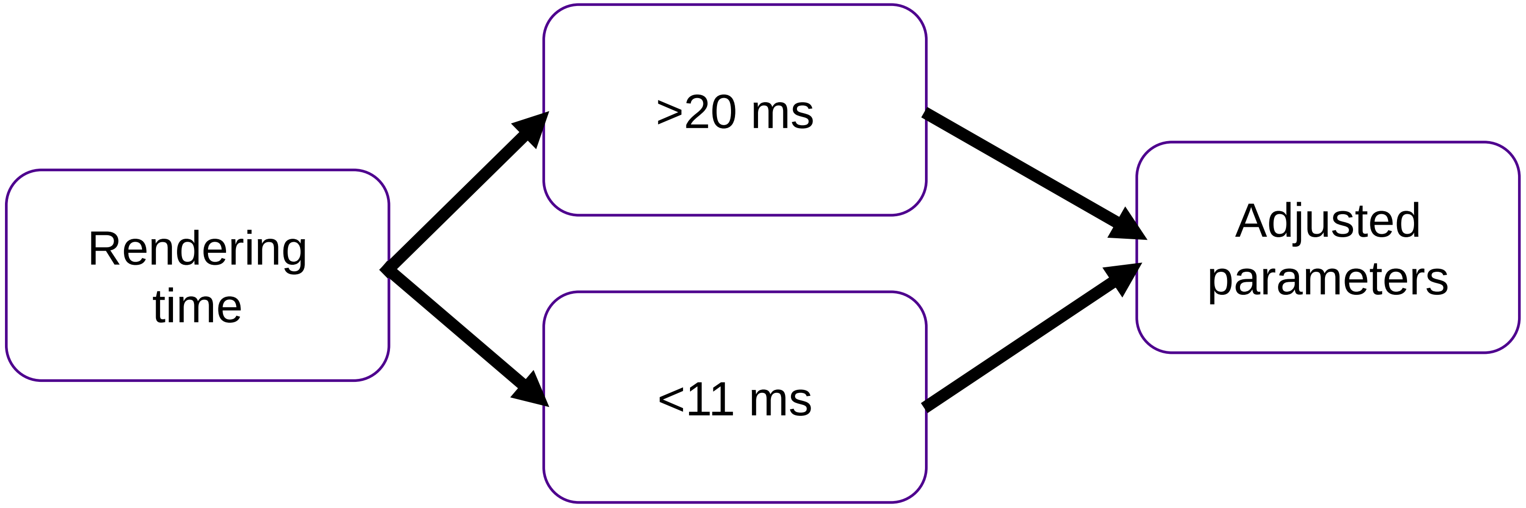
Task scheduling framework

- A **mapping algorithm** assigns **workload ratios** to the GPUs **per view**, **per stage**
- In the example below it is simply 1/N with N=4 (note: ¼ = **0.25**)
- Workload ratios are used at command buffers creation



Quality-performance control loop

- Adjusting rendering parameters (spp, bounces, SVGF iterations) while keeping the rendering time **between 11 ms and 20 ms (target range).**
- Adapt the renderer to the scene complexity



(Test scenes: San Miguel, Sponza, Bistro Exterior)

Results

Performance per stage

- Resolution: 1280x720 **per eye**
- Samples (ray) per pixel: 1
- Bounces: 6

- Stages **#0 to #7 in parallel**
- **Our proposed pipeline: stages #0 to #8 in parallel**
- **Speedup** of the main stages:
  - Spatial reprojection: **x2.75**
  - Hole-filling: **x2.89**
  - Denoising (SVGF): **x4.2**

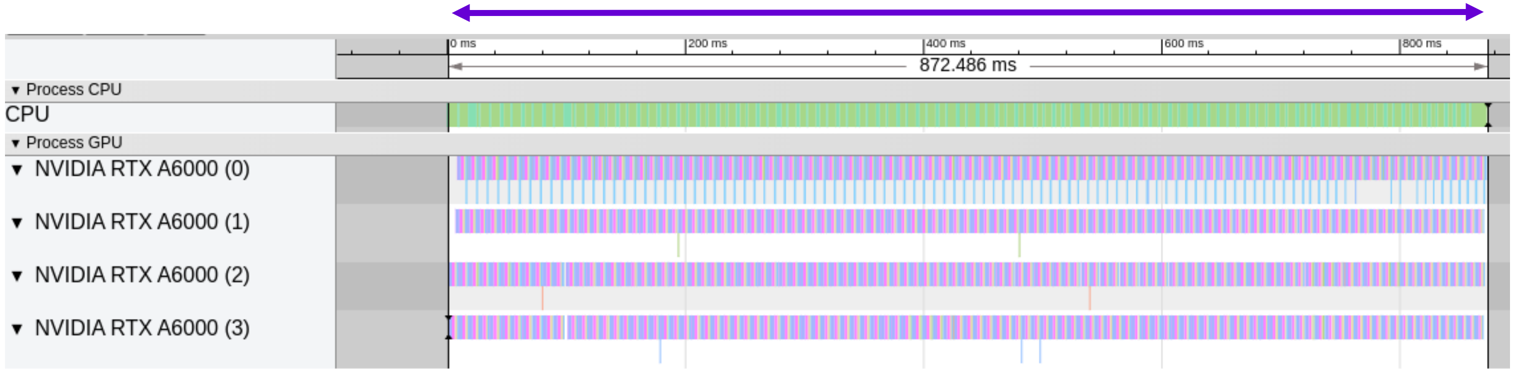
	Tauray	Proposed
Pipeline stages	Execution time (ms)	
0. Skinning	0.003	0.08
1. Scene update	0.09	0.09
2. Path tracing	3.45	3.07
3. G-buffer rasterization	2.19*	2.61
4. Spatial reprojection	0.11*	**0.04**
5. Hole-filling	3.38*	**1.17**
6. Denoising	3.65*	**0.87**
7. Record previous buffer	0.45*	0.53
8. Tonemapping	0.07*	0.03
9. Transfer to primary GPU (S+R)	1.05 + 0.60*	0.26 + 0.29*
10. Stitching image regions	0.07*	0.03*
11. Blit images to the swapchain	n/a	0.12*
	GPU utilization (%)	
Averaged utilization per GPU	99* | 53	94* | 97

(*): Values measurable only for the main GPU
Stage #9: S = Send / R = Receive

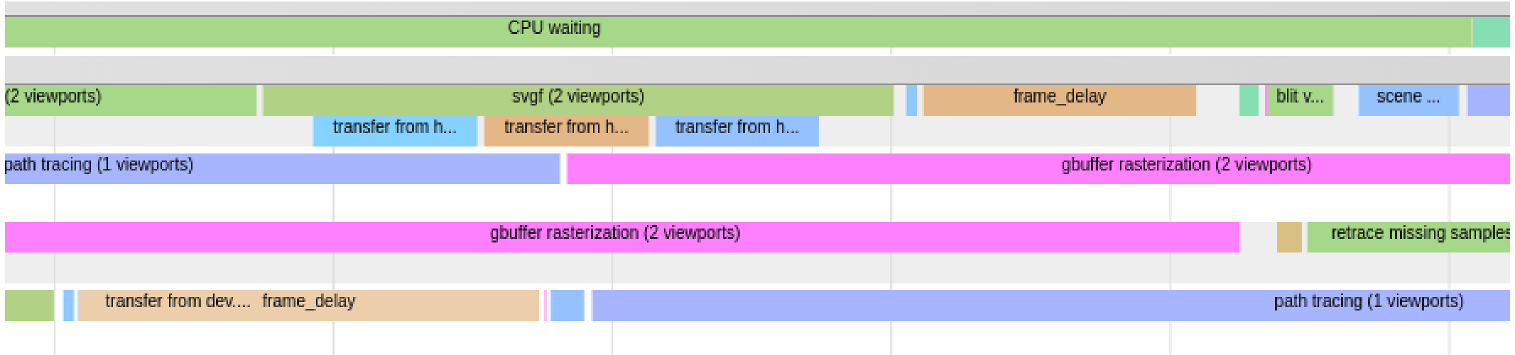
Measuring total time

- **Transfers** mainly **overlap** with the other stages
- The total rendering frequency is averaged from the **makespan (total rendering time)** and the number of frames

Makespan:
872 ms (San Miguel scene - 104 frames)



(Swimlane visualization of a real execution of the proposed pipeline)



(zoomed-in - notice overlapping stages)

Rendering frequency

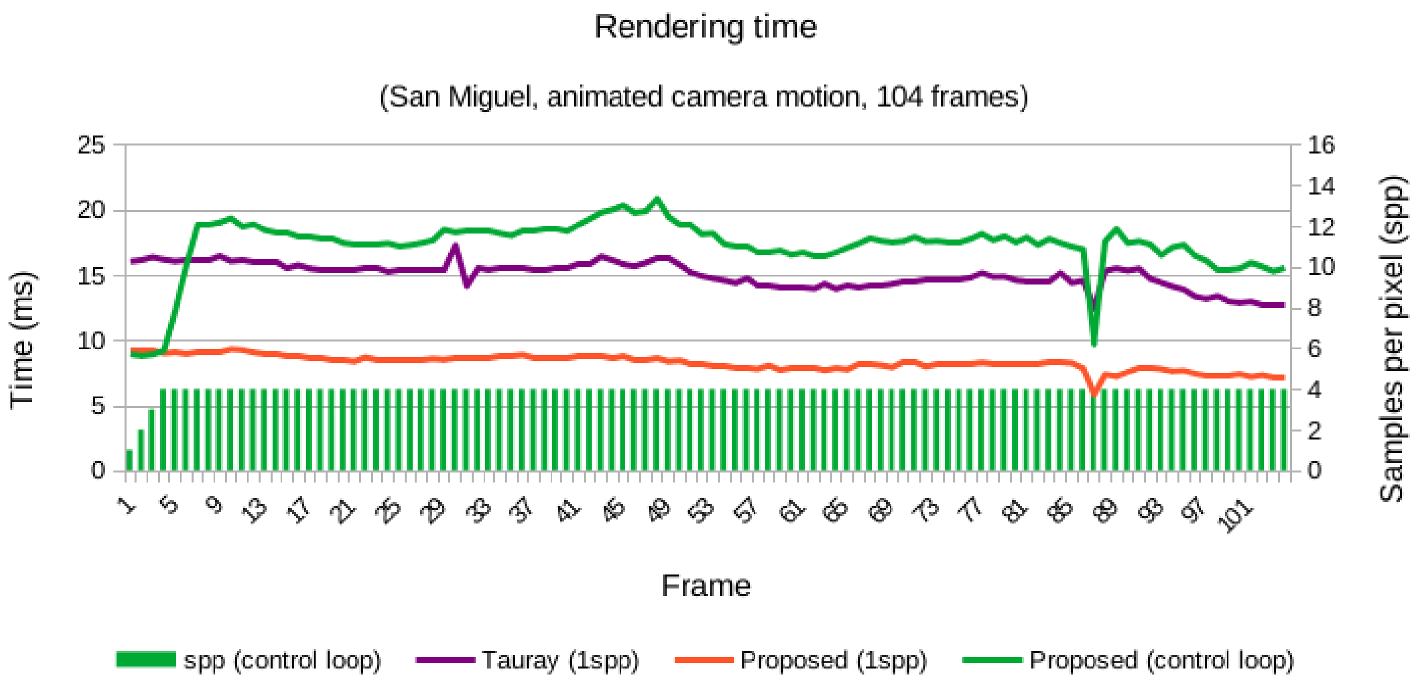
Scene	**San Miguel**	**Sponza**	**Bistro Ext.**
Pipeline	Execution time (s)		
Tauray	1.59	1.93	3.26
Proposed	**0.87**	**0.69**	**1.54**

Makespan in seconds over ~100 frames

The rendering frequency based on the **makespan's** execution time:

- San Miguel: ~120 Hz – rendering time **speedup x1.83**
- Sponza: ~145 Hz – **x2.80**
- Bistro Exterior: ~65 Hz – **x2.11**

Quality-performance control loop



- **Curves** show **rendering time** (**left axis**) for a given pipeline for each time frame
- Green **bars** indicate the **numbers of samples per pixel** (**right axis**) with the control loop
- The proposed pipeline with the control loop maintain the rendering time within the target range (11—20 ms)

Image quality related to seam artifacts

- Quality metrics **PSNR** and Contrast-Aware Multiscale Banding Index (**CAMBI**)
  - **No filter** = Non-filtered seam artifacts (proposed pipeline)
  - **Bilateral filter** = Seam artifacts smoothed with a bilateral filter (proposed pipeline)
  - **No seam** = No seam artifacts (baseline: Tauray pipeline -> **single GPU denoising**)
- PSNR difference **0.2-0.9 dB**

Scene	**San Miguel**	**Sponza**	**Bistro Ext.**
PSNR (no filter)	20.30	21.73	20.89
PSNR (bilateral filter)	19.90	21.71	20.46
PSNR (no seam)	20.83	21.97	21.09
CAMBI (no filter)	0.06	0.05	0.09
CAMBI (bilateral filter)	0.06	0.06	0.09
CAMBI (no seam)	0.07	0.06	0.09

Limitations

- **Hole-filling:** GPU warp divergence due to irregular positions of the holes in the images
- **Denoiser**: Does not consider changing number of samples per pixel + produces seam artifacts
- **Data locality**: Better data locality favors load imbalance between GPU but reduces the spatial loss of information in image-space
- **G-buffer rasterization:** slows down proportionally to the size of the scene and triangles in the view frustum

Summary

Proposed Pipeline

- Maximizes **data locality** along horizontal axis in the image
- Parallelizes spatial reprojection, hole-filling and denoising across multiple GPUs
- Handles **workload dependency** through workload ratios per GPU per stage per view
- Keeps the rendering frequency within and/or above the 50-90 Hz target range
- Tunes the quality with respect to the target rendering frequency range

Performance

- For the 3 test scenes: **x2.25** speedup for ~100 frames
- For the main stages for the San Miguel scene: **x2.75 to x4.2** speedup

Quality

- No significant degradation due to seam artifacts

Authors (first.last@tuni.fi)


Erwan Leria (Presenter)


Markku Mäkitalo


Pekka Jääskeläinen

Virtual Reality and Graphics Architectures research group (VGA)
tuni.fi/vga/

This work was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 956770 (Plenoptima), and in part by the Academy of Finland under Grant 325530 and Grant 351623